

## PATENT APPLICATION

### METHOD AND SYSTEM FOR PROVIDING SECURE REMOTE ACCESS TO EMAIL RESOURCES ON PRIVATE NETWORKS

- Inventor(s):
1. Theron Tock  
620 Palo Alto Ave.  
Mountain View, CA 94041  
Citizenship: USA
  2. Sampath Srinivas  
3443 Georgetown Place  
Santa Clara, CA 95051  
Citizenship: USA
  3. Yvonne Pui Fung Sang  
20453 Summercrest Dr.  
Castro Valley, CA 94552

Assignee: Neoteris, Inc.

# **METHOD AND SYSTEM FOR PROVIDING SECURE REMOTE ACCESS TO EMAIL RESOURCES ON PRIVATE NETWORKS**

## **CROSS-REFERENCE TO RELATED APPLICATIONS**

[0001] This application claims priority of U.S. Provisional Patent Application No. 60/\_\_\_\_\_ (Att.Dkt.No. DANAP005P), filed November 2, 2001, and entitled "METHOD AND SYSTEM FOR PROVIDING REMOTE SECURE ACCESS TO PRIVATE NETWORKS," which is hereby incorporated herein by reference.

[0002] In addition, this application is related to (i) U.S. Patent Application No. \_\_\_\_\_ (Att.Dkt.No. DANAP005), filed concurrently herewith, and entitled "METHOD AND SYSTEM FOR PROVIDING SECURE ACCESS TO PRIVATE NETWORKS," which is hereby incorporated herein by reference, and (ii) U.S. Patent Application No. 09/706,181, filed November 3, 2000, and entitled "METHOD AND SYSTEM FOR REQUESTING AND PROVIDING CONTENT FROM SERVER TO CLIENT VIA AN INTERMEDIARY SERVER," which is hereby incorporated herein by reference. This application is also related to U.S. Patent Application Nos. 09/706,182; 09/706,296; and 09/706,297.

## **BACKGROUND OF THE INVENTION**

### **Field of the Invention**

[0003] The present invention relates to client-server computing and, more particularly, to client-server computing for securely accessing resources over a network.

### **Description of the Related Art**

[0004] Network browsers (browser applications), such as Netscape Navigator or Microsoft Explorer, allow users of client machines to request and retrieve resources from remotely located server machines via the Internet. These network browsers can display or render HyperText Markup Language (HTML) documents provided by the remotely located server machines. Additionally, browsers are able to execute script programs embedded in the HTML documents to provide some local functionality.

[0005] Conventionally, network browsers are used to access public networks, such as the Internet. Private networks are normally protected by firewalls so that network browsers residing on computing machines outside the private network are not able to gain access to any resources on the private network.

[0006] While firewalls are effective at protecting against external access to private networks, there is often the need for external persons or businesses to gain at least limited access to the private networks of other persons or businesses. For example, a supplier of parts to a business customer may be able to better serve their business customer by having access to information (e.g., inventory levels or orders) maintained on the private network of the business customer. One conventional approach is to allow the supplier's machine to access the private network through the firewall via a public network. This provides a "hole" in the firewall that seriously compromises the security of the private network. Hence, this conventional approach is normally not permitted if security is an important concern. Another conventional approach is to establish a Virtual Private Network (VPN) with the supplier's machine. Here, the supplier's machine is also able to access the private network through the public network and the firewall, but all data transmissions are encrypted. Some firewalls support VPNs and protocols providing the encrypted communications, such as Point-to-Point Tunneling Protocol (PPTP), can be used. While VPNs offer remote secure access, they are difficult to arrange, configure and manage. Each VPN must also be provided for each external person or business given access to the private network. Still further VPNs are costly and each VPN provides some security exposure to the entire private network.

[0007] Thus, there is a need for improved approaches to providing secure remote access to resources maintained on private networks.

### **SUMMARY OF THE INVENTION**

[0008] The invention pertains to improved approaches for providing secure remote access to email resources maintained on private networks. The secure access can be provided through a public network using a standard network browser. Multiple remote users are able to gain restricted and controlled access to email on a mail server within a private network through a common access point. The solution

provided by the invention allows not only native access to email resources but also robust authentication approaches.

[0009] The invention can be implemented in numerous ways, including as a system, method, device, and a computer readable medium. Several embodiments of the invention are discussed below.

[0010] As a method for facilitating remote access by a mail client to a mail server via an intermediary server, one embodiment of the invention includes at least the acts of: receiving a mail access request at the intermediary server, the mail access request being sent to the intermediary server from the mail client for a requestor; receiving a password associated with the mail access request; authenticating the requestor with the mail server based on the received password; authenticating the requestor with an authentication server based on the received password, the authentication server being associated with a private network that includes the mail server; and permitting the mail access request when both the mail server and the authentication server authenticate the requestor.

[0011] As a method for authenticating a requestor of a remote mail client seeking access to a mail server, one embodiment of the invention includes at least the acts of: receiving a password from the remote mail client; retrieving a previously stored hashed password; determining whether a hashed version of the received password matches the previously stored hashed password; authenticating the requestor with the mail server based on the received password; and further authenticating the requestor with an authentication server based on the received password when it is determined that the hashed version of the received password does not match the previously stored hashed password, the authentication server being associated with a private network that includes the mail server.

[0012] As a computer readable medium including at least computer program code for facilitating remote access by a mail client to a mail server via an intermediary server, one embodiment of the invention includes at least: computer program code for receiving a mail access request at the intermediary server, the mail access request being sent to the intermediary server from the mail client for a requestor; computer program code for receiving a password associated with the mail access request; computer program code for authenticating the requestor with the mail server based on the received password; computer program code for

authenticating the requestor with an authentication server based on the received password, the authentication server being associated with a private network that includes the mail server; and computer program code for permitting the mail access request when both the mail server and the authentication server authenticate the requestor.

[0013] As a computer readable medium including at least computer program code for authenticating a requestor of a remote mail client seeking access to a mail server, one embodiment of the invention includes at least: computer program code for receiving a password from the remote mail client; computer program code for retrieving a previously stored hashed password; computer program code for determining whether a hashed version of the received password matches the previously stored hashed password; computer program code for authenticating the requestor with the mail server based on the received password; and computer program code for authenticating the requestor with an authentication server based on the received password when the computer program code for determining determines that the hashed version of the received password does not match the previously stored hashed password, the authentication server being associated with a private network that includes the mail server.

[0014] Other aspects and advantages of the invention will become apparent from the following detailed description taken in conjunction with the accompanying drawings which illustrate, by way of example, the principles of the invention.

### **BRIEF DESCRIPTION OF THE DRAWINGS**

[0015] The invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

FIG. 1A is a block diagram of an information retrieval system according to one embodiment of the invention.

FIG. 1B is a block diagram of an information retrieval system according to another embodiment of the invention.

FIG. 2A is a block diagram of an intermediary server according to one embodiment of the invention.

FIG. 2B is a block diagram of a remote access system according to one embodiment of the invention.

FIG. 3 is a flow diagram of request processing according to one embodiment of the invention.

FIG. 4 is a flow diagram of authentication processing according to one embodiment of the invention.

FIG. 5 is a flow diagram of access privilege processing according to one embodiment of the invention.

FIG. 6 is a flow diagram of operational privilege processing according to one embodiment of the invention.

FIG. 7 is a flow diagram of detailed external authentication processing according to one embodiment of the invention.

FIGs. 8A and 8B are flow diagrams of file access request processing according to one embodiment of the invention.

FIGs. 9A – 9C are flow diagrams of web resource request processing according to one embodiment of the invention.

FIG. 10 illustrates a diagram of an information retrieval system according to one embodiment of the invention.

FIG. 11 is a flow diagram of URL modification processing according to one embodiment of the invention.

FIG. 12 is a flow diagram of script modification processing according to one embodiment of the invention.

FIGs. 13A and 13B are flow diagrams of script modification processing according to another embodiment of the invention.

FIG. 14 is a flow diagram of email request processing according to one embodiment of the invention.

FIG. 15 is a flow diagram of mail operation processing according to one embodiment of the invention.

FIG. 16 is a flow diagram of authentication processing according to one embodiment of the invention.

FIGs. 17A and 17B illustrate an example of a computer system that may be used in accordance with the invention.

## DETAILED DESCRIPTION OF THE INVENTION

[0016] The invention pertains to improved approaches for providing secure remote access to email resources maintained on private networks. The secure access can be provided through a public network using a standard network browser. Multiple remote users are able to gain restricted and controlled access to email on a mail server within a private network through a common access point. The solution provided by the invention allows not only native access to email resources but also robust authentication approaches.

[0017] Embodiments of this aspect of the invention are discussed below with reference to FIGs. 1A – 17B. However, those skilled in the art will readily appreciate that the detailed description given herein with respect to these figures is for explanatory purposes as the invention extends beyond these limited embodiments.

[0018] FIG. 1A is a block diagram of an information retrieval system 100 according to one embodiment of the invention. The information retrieval system 100 includes a network 102, client machines 104 and 106, an intermediary server 108, remote servers 110 and 112, a private network 114, and private servers 116 and 118. The network 102 serves as a communication medium through which the client machines 104 and 106, the intermediary server 108 and the remote servers 110 and 112 can communicate. The network 102 is, for example, a data network which can include the Internet, a wide area network, or a local area network. The Internet refers to a global network of interconnected computers. The private network 114 also serves as a communication medium through which the intermediary server 108 and the private servers 116 and 118 can communicate. The network 114 is also a data network. Often the private network 114 is associated with an entity and thus employees operating computing devices on the private network 114 are able to communicate with the private servers 116 and 118. For example, the private network 114 can be referred to as a corporate network or an intranet. However, access to the private network 114 by an outside computing device is typically limited by a firewall (not shown). The intermediary server 108 is permitted to communicate with the private network 114 through the firewall. Here, to the extent a client machine (requestor) is authorized and permitted, the intermediary server 108

communicates with the private network 114 on behalf of the client machine (requestor). The intermediary server 108, in effect, controls the extent to which it allows outside computing devices to access the private network 114.

[0019] According to the invention, requests for content residing on the private servers 116 and 118 can be received from the client machines 104 and 106. As used herein, "content" is any information or resource that can be stored on a server and retrieved by a client. Typically, the content is embodied as an electronic file and contains text and/or images. Often, the client machines 104 and 106 operate browser applications that facilitate requesting and retrieval of content over the network 102 and the private network 114. In such cases, the content is often returned to the browser application as a browser-viewable document (e.g., markup language document, webpage, etc.) so that the browser application can display the same. The client machines 104 and 106 communicate with an intermediary server 108. Initially, the intermediary server 108 determines whether the client machines 104 and 106 seeking the content are authenticated and permitted such access to the private network 114. Following successful authentication and permission verifications, the intermediary server 108 then, in turn, accesses the private servers 116 and 118 residing on the private network 114 on behalf of the client machines 104 and 106. Once the intermediary server 108 obtains the requested content from the private servers 116 and 118, the intermediary server 108 can directly return the requested content to the client machines 104 and 106 or can first modify the requested content and then deliver it to the client machines 104 and 106.

[0020] The modification to the requested content by the intermediary server 108 can take a variety of forms. As one example, the intermediary server 108 can insert a toolbar into the requested content before delivery to the client machines 104 and 106. As another example, the intermediary server 108 can alter the hyperlinks within the requested content so as to point to an intermediary server (e.g., the intermediary server 108). Various other tasks can be performed on the requested content by the intermediary server 108. Additionally, the information retrieval system 100 can also support centralized storage at the intermediary server 108 of server stored information. The server stored information is often referred to as "cookies," though cookies are conventionally stored on client machines.



[0021] Although the information retrieval system 100 illustrated in FIG. 1A depicts only a pair of client machines, a pair of remote servers, a single intermediary server and a pair of private servers, it should be understood that the information retrieval system 100 can support many client machines and many server machines. It should also be understood that the information retrieval system 100 can also support multiple intermediary servers.

[0022] FIG. 1B is a block diagram of an information retrieval system 150 according to one embodiment of the invention. The information retrieval system 150 is, for example, a more detailed implementation of the information retrieval system 100 illustrated in FIG. 1A.

[0023] The information retrieval system 150 makes use of the Internet 152 and client machines 154 and 156 that couple to the Internet 152 through wired or wireless means. Typically, the client machines 154 and 156 operate client-side applications, such as a network browser or a mail application. When requestors (users) of the client machines 154 and 156 desire to access remote resources, resource requests are sent from the client machines 154 and 156 through the Internet 152 to an intermediary server 158. Typically, the communications between the client machines 154 and 156 and the intermediary server 158 are secured by an encryption technique (e.g., Secure Socket Layer (SSL)). The intermediary server 158 provides access to an intranet 160. The resources being requested by the client machines 154 and 156 reside within the intranet 160. Since a firewall typically limits or precludes external access to the intranet 160, the intermediary server 158 must be permitted to communicate with the intranet through the firewall 162. The intranet 160 typically includes various different types of resources that can be accessed electronically. Typically, these resources are stored on server machines that couple to, or form part of, the intranet. As shown in FIG. 1B, the intranet 160 couples to, or includes, an authentication server 164, a web server 166, a mail server 168, a file server 170 and a log server 172. Hence, a given client machine can access any one of the servers 164-172 residing within or on the intranet 160 by way of the intermediary server 158. Consequently, a given client machine can request and receive resources residing on the web server 166 using a network browser application. As another example, the given client machine can access the mail resources residing on the mail server 168 using a client-side mail application.

As still another example, the given client machine can access the file server 170 residing within or on the intranet 160 to obtain, store or view electronic files thereon.

[0024] The intermediary server 158 is configured to ensure that access to the intranet 160 via the intermediary server 158 remains protected. In this regard, the requestors that are seeking to access resources or content residing on the intranet 160 must be authenticated. The authentication can utilize the authentication server 164 residing within or on the intranet 160. In this regard, native authentication techniques utilized by the intranet 160 can be used in authenticating a requestor at the intermediary server 158. Still further, the intermediary server 158 can be configured by an administrator such that different requestors (e.g., users of client machines) can be given different access privileges to different resources (e.g., servers) within or on the intranet 160. The log server 172 allows the storage of log information pertaining to access requests to the intranet 160 at the intermediary server 158. The log information can be provided on an application level basis such that it is more user-discernable.

[0025] FIG. 2A is a block diagram of an intermediary server 200 according to one embodiment of the invention. The intermediary server 200 is, for example, suitable for use as the intermediary server 108 illustrated in FIG. 1.

[0026] The intermediary server 200 includes various processing modules typically implemented by computer program code executed by a processing device utilized by the intermediary server. More particularly, the processing modules of the intermediary server 200 include a web server 202 and a protocol handler 204. The web server 202 couples to client machines through a link 206 (via a network) and the protocol handler 204 couples to remote servers through a link 208 (via a network). The web server 202 and the protocol handler 204 also communicate with one another as well as with various supporting modules and a data storage device 210. The data storage device 210 provides persistent or non-volatile storage for various data items being maintained by the intermediary server 200. Typically, for each user or requestor associated with a client machine, the data storage device provides separate storage.

[0027] The processing modules include an authentication manager 212 and an access manager 214. The authentication manager 212 manages authentication processing which serves to determine whether the requestor is who they say they

are. The authentication processing can be local or external to the intermediary server 200. For example, external authentication can be provided by an authentication server within a private network (e.g., authentication server 164). The access manager 214 provides access limitations to the various resources on the private network. Namely, different requestors can be assigned different levels, types or areas of access privileges. For example, requestor A can access server X but not servers Y and Z, and requestor B can access server Y for read-only and not servers X and Z.

[0028] The intermediary server 200 also includes a content transformer 216, which is another processing module that is used to parse requested content received from a remote server and then modify the content in predetermined ways.

[0029] Another processing module that the intermediary server 200 might include is a cookie manager 218. The cookie manager manages "cookies" such that those being received from a remote server are stored to the data storage device 210 and those "cookies" previously stored in the data storage device 210 are delivered to the remote server at appropriate times. More generally, "cookies" refer to server stored information. Such server stored information is often set by a remote server and used for session, state or identification purposes.

[0030] FIG. 2B is a block diagram of a remote access system 250 according to one embodiment of the invention. The remote access system 250 operates in a client-server environment to allow users of clients to gain access to resources at remote servers. In particular, the remote access system 250 includes a network browser 254 and a mail client 256. The network browser 254 and the mail client 256 are client applications that operate or run on client machines. Typically, a user or requestor will interact with these one or more client programs to request resources located on the remote servers. The network browser 254 and the mail client 256 couple to an intermediary server 252 over secure links or connections. The intermediary server 252 also couples to the remote servers through either secure or unsecure connections or links. The intermediary server 252 can support connections to various different servers, such as servers found on private networks. One example of a private network is a corporate network. The servers illustrated in FIG. 2B include a web server 258, an email server 260, a Windows file server 262, a UNIX file server 264, an authentication server 266 and a log server 268.

[0031] The intermediary server 252 includes a Secure Socket Layer (SSL) 272 that provides encryption handling for the connection or link with the client applications prior to reaching a front-end protocol handler layer 270. The front-end protocol handler layer 270 includes a plurality of protocol handlers to handle the different types of incoming protocols that may be utilized by the various client applications. As shown in FIG. 2B, the front-end protocol handler layer 270 includes separate protocol handlers for the protocols of HTTP, IMAP, SMTP, POP, and MAPI. After the appropriate protocol handler has been utilized for an incoming request, other functional modules within the intermediary server 252 can then be utilized. In particular, an access manager 274 can determine whether the requestor associated with the incoming request is permitted the type of access being requested. An authentication manager 276 can determine whether the requestor is able to be authenticated. A content transformer 278 can perform transformation of the content of the received request or the requested response provided by the remote server. A system administration manager 280 allows a system administrator to interact with the intermediary server 252 to configure access privileges, system configuration and login features.

[0032] The intermediary server 252 also includes back-end protocol handlers 282. The back-end protocol handlers 282 provide the appropriate protocol for outgoing and incoming communications with respect to a particular server. The layer of back-end protocol handlers shown in FIG. 2B includes protocol handlers for the protocols of: HTTP, IMAP, SMTP, POP, SMB, NFS, NIS, RADIUS, LDAP, and NT. To the extent that an incoming protocol to the intermediary server 252 differs from an outgoing protocol from the intermediary server 252, the content transformer 278 can perform the protocol transformations (e.g., translations). Still further, the intermediary server 252 includes a data store 284, a log manager 286, and a data synchronization manager 288. The data store 284 can provide temporary or semi-permanent data storage for the various components of the intermediary server 252. For example, a local record for authentication purposes can be stored for each of the clients or requestors in the data store 284. In addition, session identifiers, or cookies, for the clients or requestors can also be stored in a centralized fashion in the data store 284. The data synchronization manager 288 is a module that enables coupling of one intermediary server with another intermediary server to provide fault

tolerance. Hence, if one intermediary server fails, then, through a link 290, the failing intermediary server can couple to an operating intermediary server to provide some or all of the operations typically associated with an intermediary server. The log manager 286 is provided to enable application level logging of various access requests that are made through the intermediary server 252. The log formed by the log manager 286 is stored in the log server 268.

[0033] FIG. 3 is a flow diagram of request processing 300 according to one embodiment of the invention. The request processing 300 is invoked whenever a request from a requestor is received by an intermediary server, such as the intermediary server 108 illustrated in FIG. 1A, the intermediary server 158 illustrated in FIG. 1B, the intermediary server 200 illustrated in FIG. 2A or the intermediary server 252 illustrated in FIG. 2B.

[0034] The request processing 300 begins with a decision 302 that determines whether the received request is a system login request. When the decision 302 determines that the received request is a system login request, then the request processing 300 attempts to authenticate 304 the requestor. The authentication can be performed locally or remotely. Additional details on authentication are provided below. Thereafter, a decision 306 determines whether the requestor has been authenticated. When the decision 306 determines that the requestor cannot be authenticated, then the login attempt fails and a login page can be returned 308 to the requestor. The login page facilitates login retry by the requestor. Following the operation 308, the request processing 300 is complete and ends for the case in which the login request failed.

[0035] Alternatively, when the decision 306 determines that the requestor is authenticated, then a session identifier is returned 310 to the requestor. The requestor can refer to a client device or the user of the client device depending on context. The session identifier is used in subsequent requests to the intermediary server as long as the session is active. Additionally, an initial access page is returned 312 to the requestor. From the initial access page, the requestor is able to access various resources available on a private network. Following the operation 312, the request processing 300 is complete and ends for the case in which the login request was successful.

[0036] Besides the processing of login requests, the request processing 300 also operates to process all other requests for remote access via the intermediary server. Hence, when the decision 302 determines that the received request is not a system login request, then a decision 314 determines whether the received request has a valid session identifier. The received request would have a valid session identifier if the requestor has already been authenticated (i.e., logged into the intermediary server) and the session is still valid. Hence, when the decision 314 determines that the session identifier associated with the received request is not valid, then access to the intermediary server is denied and the login page can be returned 308 to the requestor. Alternatively, when the decision 314 determines that the session identifier is valid, then a decision 316 determines whether the session has timed-out. When the decision 316 determines that the session has timed-out, then access to the intermediary server is denied and the login page can be returned 308 to the requestor. Here, if the requestor has an invalid session identifier or the session has timed-out, the requestor is forced to login to be authenticated.

[0037] On the other hand, when the decision 316 determines that the session has not timed-out, then the requestor is permitted to access the private network via the intermediary server. Thereafter, depending upon the type of access the requestor is seeking to make, additional processing is performed to ensure that the requestor gains access to only those resources deemed appropriate and intended. In particular, with respect to the request processing 300, access privileges associated with the requestor are obtained 318. The access privileges indicate which resources the requestor is entitled to access. Next, a decision 320 determines whether the particular access type associated with the received request is permitted. When the decision 320 determines that the access type associated with the received request is not permitted, then an access denied page is returned 322 to the requestor. Alternatively, when the decision 320 determines that the access type of the received request is permitted, then the received request is permitted 324 to be processed. Here, the processing of the received request enables the requestor to access (e.g., view, retrieve, etc.) the protected resources from the private network. Following the operations 322 and 324, the request processing 300 is complete and ends with the received request having been processed only when access is deemed permitted.

[0038] FIG. 4 is a flow diagram of authentication processing 400 according to one embodiment of the invention. The authentication processing 400 is, for example, processing associated with the authentication operation 304 illustrated in FIG. 3.

[0039] The authentication processing 400 begins with a decision 402 that determines whether a local record for the requestor (user) exists. When the decision 402 determines that a local record for the requestor does exist, then a decision 404 determines whether local or external authentication is required. Here, the local record indicates whether local or external authentication should be performed. Besides an indication of whether local or external authentication should be performed, a local record can also store other useful information, for example, requestor's (user's) name, time last logged in, account status, etc. When the decision 404 determines that local authentication is to be performed, a password provided with the login request being processed for authentication is hashed 406. Hashing is the transformation of a string of characters into another string of characters referred to as a "key" that represents the original string. A hash function can perform the hashing operation. Hashing is often performed in the encryption and decryption context.

[0040] Next, a decision 408 determines whether the hashed password matches a stored hash password. When the decision 408 determines that a match is present, then the authentication is deemed successful 410. Alternatively, when the decision 408 determines that a match is not present, then the authentication is deemed to have failed 412. Further, when the decision 408 determines that there is no match, then an access failure can also be logged 414 in a log. In one embodiment, the log can be provided by a log server. The logging 414 of the access failure can provide application-level information that facilitates understanding of the nature of the access failure that occurred when later viewing the log. Following the operations 410 and 414, the authentication processing 400 is complete and ends with the authentication either succeeding or failing depending on whether the login request contains the correct password.

[0041] On the other hand, when the decision 402 determines that a local record for the requestor does not exist, then a decision 416 determines whether a local setting is required. A system setting can be used to indicate whether or not a local

record is required. An administrator can use such a system setting to limit access to only those users having local records. When the decision 416 determines that a local setting is required, then the authentication is deemed to have failed 412 because there is no available local record. Again, the access failure can be logged 414. Alternatively, when the decision 416 determines that a local setting is not required, or when the decision 404 determines that external authentication is to be performed, then an address and type of external authentication server (EAS) to be used for the authentication are obtained 418. Different processing is typically performed with different types of external authentication servers. Normally, these external authentication servers are already provided within the private network for purposes of performing authentications. Typically, there is a system setting that indicates a particular external authentication server to be used. Hence, the authentication processing 400 can make use of the native authentication provided by such external authentication servers. The discussion below pertaining to FIG. 7 provides additional detail on different types of external authentications.

[0042] Next, a decision 420 determines whether the external authentication has been successful. Here, external authentication is performed depending upon the particular type of external authentication that has been indicated. When the decision 420 determines that external authentication is not successful, then the authentication is deemed to have failed 412. Additionally, the access failure can be logged 414 as previously discussed. On the other hand, when the decision 420 determines that the external authentication has been successful, then the authentication is deemed to be successful 422. Following the operation 422, the authentication processing 400 is complete and ends with the requestor being authenticated.

[0043] FIG. 5 is a flow diagram of access privilege processing 500 according to one embodiment of the invention. The access privilege processing 500 is, for example, processing performed by the decision 320 of FIG. 3. Namely, the access privilege processing 500 determines whether the access type being requested is permitted by a particular requestor. In effect, the access type provides various criteria that can be used to limit access by requestors. With respect to the embodiment shown in FIG. 5, the criteria includes source Internet Protocol (IP) address, time-of-day, and operations.



[0044] The access privilege processing 500 begins with a decision 502 that determines whether the source IP address associated with the received request (i.e., the requestor) is authorized. When the decision 502 determines that the source IP address associated with the received request is not authorized, then the access privilege processing 500 denies access 504. Here, to reduce risk of unauthorized access, the access privilege processing 500 ensures that only those IP addresses of known requestors are able to access the private resources.

[0045] When the decision 502 determines that the source IP address is authorized, then a decision 506 determines whether the time at which the request is being made satisfies a time-of-day access limitation. Typically, this limitation can be configured for all requestors or separately for each requestor. Here, the intermediary server can be configured, if desired, to permit access to private resources only during certain time periods. This, for example, can permit access only during business hours or other limited hours. When the decision 506 determines that the time of the received request is not within the time-of-day permitted, then the access privilege processing 500 denies access 504.

[0046] When the time associated with the received request is determined 506 to be within the time-of-day permitted, a decision 508 determines whether the particular operation associated with the received request is permitted. Here, the incoming request can request various different operations to be performed with respect to the private resources. These various different operations tend to vary with type of application being provided. The decision 508 can operate to limit the operations permitted to be used by different requestors. When the decision 508 determines that the operation being requested is not permitted, then access is denied 504. On the other hand, when the decision 508 determines that the requested operation is permitted, then access is permitted 510. Following the operations 504 and 510, the access privilege processing 500 is complete and ends.

[0047] FIG. 6 is a flow diagram of operational privilege processing 600 according to one embodiment of the invention. The operational privilege processing 600 is, for example, performed by the decision 508 illustrated in FIG. 5. It should also be noted that the operational privilege processing 600 performs the requested operation when such operation is determined to be permitted, and thus can be associated with the operations 320 and 324 of FIG. 3.

[0048] The operational privilege processing 600 begins with a decision 602 that determines whether a file browsing operation has been requested. When the decision 602 determines that a file browsing operation has been requested, then a decision 604 determines whether file browsing is enabled for the requestor. When the decision 604 determines that file browsing is not enabled for the requestor, then access is denied 606 and thus the operational privilege processing 600 ends. Alternatively, when the decision 604 determines that file browsing is enabled for the requestor, then a decision 608 determines whether a read or write operation is being requested. When the decision 608 determines that a write operation is requested, a decision 610 determines whether write access is permitted. In one embodiment, the decision 610 determines whether write access is permitted by the particular requestor making the request. When the decision 610 determines that write access is not permitted, then access is denied 606 and thus the operational privilege processing 600 ends. Alternatively, when the decision 610 determines that write access is permitted, then write request processing is performed 612 to carry out the received request. Following the operation 612, the operational privilege processing 600 ends with the requested operation having been performed.

[0049] On the other hand, when the decision 608 determines that a read operation is being requested, a decision 614 determines whether read access is permitted. In one embodiment, the decision 614 determines whether read access is permitted by the particular requestor making the request. When the decision 614 determines that read access is not permitted, then access is denied 606. Alternatively, when the decision 614 determines that read access is permitted, then read request processing is performed 616 to carry out the requested operation. Following the operation 616, the operational privilege processing 600 is complete and ends with the requested operation having been performed.

[0050] On the other hand, when the decision 602 determines that the requested operation is not a file browsing operation, a decision 618 determines whether the requested operation is a web browsing operation. When the decision 618 determines that the requested operation is a web browsing operation, a decision 620 determines whether the server associated with the web browsing operation is accessible to the requestor. When the decision 620 determines that the server is not accessible to the requestor, then access is denied 606. In one embodiment, the

intermediary server can maintain a list of servers that are accessible by particular requestors. This enables the intermediary server to control the resources that particular requestors are able to browse by server names. For example, although a private network may include numerous servers, requestors are able to be individually restricted to accessing only certain servers. Alternatively, when the decision 620 determines that the server associated with the web browsing operation is accessible to the requestor, then the web browsing request processing is performed 622. In other words, the requested web browsing operation is performed 622 because the requestor was entitled to access the particular server. Following the operation 622, the operational privilege processing 600 ends with the requested operation having been performed.

[0051] On the other hand, when the decision 618 determines that the requested operation is not a web browsing operation, then a decision 624 determines whether the requested operation is an email operation. When the decision 624 determines that the requested operation is an email operation, then a decision 626 determines whether email (electronic mail) is enabled for the requestor. When the decision 626 determines that email is not enabled for the requestor, then access is denied 606. Here, the intermediary server is able to control access to email operations by particular requestors. Alternatively, when the decision 626 determines that email is enabled for the requestor, the email request processing is performed 628. In other words, the requested email operation is performed because the requestor had suitable privileges to perform the operation. Following the operation 628, the operational privilege processing 600 ends with the requested operation having been performed.

[0052] Still further, when the decision 624 determines that the requested operation is not an email operation, then a decision 630 determines whether the requested operation is some other operation that is permitted by the intermediary server. Here, the other operation can be any suitable operation that is facilitated by the intermediary server. In effect, the other operation can represent a generic operation that is available on the intermediary server. The other operation can also refer to a local operation being performed by the intermediary server without access to a private network. Examples of local operations can vary widely but can include: adding bookmarks, adding, editing or deleting local records, altering file shares, etc.

However, the other operation could also be an operation performed within the private network. When the decision 630 determines that the requested operation is one of the other operations, then a decision 632 determines whether the other operation is permitted. When the decision 632 determines that the requested operation is not one of the other operations that are permitted, then access is denied 606. Alternatively, when the decision 632 determines that the other operation is permitted, then the other request processing is performed 634. Following the operation 634, the operational privilege processing 600 is complete and ends with the other type of operation having been performed.

[0053] On the other hand, when the decision 630 determines that the requested operation is not one of the other operations that are permitted (by the requestor), then the operational privilege processing 600 ends without having performed the requested operation. Here, since the requested operation was unsupported by the operational privilege processing 600, the requested operation is not processed (i.e., it is blocked) at the intermediary server.

[0054] FIG. 7 is a flow diagram of detailed external authentication processing 700 according to one embodiment of the invention. The detailed external authentication processing 700 is, for example, detailed processing associated with the decision 420 illustrated in FIG. 4. The detailed external authentication processing 700 supports a variety of different types of external authentication systems, including: Network Information System (NIS), Remote Authentication Dial-In User Service (RADIUS), Lightweight Directory Access Protocol (LDAP), and NT domain. Hence, the external authentication performed for the intermediary server can use any of a variety of native authentication approaches that a private network might provide.

[0055] The detailed external authentication processing 700 begins with a decision 702 that determines whether the external authentication server (EAS) is NIS. When the external authentication server is NIS, then a NIS record is read 704. Then, the password provided with the login request is hashed 706. The hashed password is compared 708 with that provided within the NIS record. A decision 710 then determines whether the hashed passwords match. When the passwords do match, the authentication succeeds 712. When the passwords do not match, the authentication fails 714.

[0056] On the other hand, when the decision 702 determines that the external authentication server is not NIS, a decision 716 determines whether the external authentication server is RADIUS. When the external authentication server is RADIUS, then the username and password provided with the login request are encrypted 718 using a RADIUS shared secret. The RADIUS shared secret is typically a shared key. Then, the encrypted value is sent 720 to the RADIUS server for authentication. A decision 722 then determines whether a response from the RADIUS server has been received. The response, when received, indicates 724 success or failure of the authentication.

[0057] On the other hand, when the decision 716 determines that the external authentication server is not RADIUS, then a decision 726 determines whether the external authentication server is LDAP. When the decision 726 determines that the external authentication server is LDAP, the username and password provided with the login request are sent 728 to the LDAP server for authentication. A decision 730 then determines whether a response from the LDAP server has been received. The response, when received, indicates 732 success or failure of the authentication.

[0058] On the other hand, when the decision 726 determines that the external authentication server is not LDAP, a decision 734 determines whether the external authentication server is NT domain (NT domain server). When the decision 734 determines that the external authentication server is NT domain, a random number is obtained 736 from the NT domain server. Then the password provided with the login request is hashed 738 with the random number. Next, the hashed value is sent 740 to the NT domain server for authentication. A decision 742 then determines whether a response from the NT domain server has been received. The response indicates 744 success or failure of the authentication.

[0059] FIGs. 8A and 8B are flow diagrams of file access request processing 800 according to one embodiment of the invention. The file access request processing 800 is, for example, the processing performed when a web browsing operation has been requested by a requestor. In other words, the file access request processing 800 can, for example, be the processing performed by one embodiment of the block 622 of FIG. 6.

[0060] The file access request processing 800 begins with a decision 802 that determines whether a server has been discovered. When the decision 802

determines that a server has already been discovered, then a decision 804 determines whether the file access request seeks to view folder contents. When the decision 804 determines that the file access request does desire to view folder contents, then the content of the folder is retrieved 806. The retrieved content is then sent 808 to the requestor.

[0061] On the other hand, when the decision 804 determines that the file access request does not seek to view folder contents, a decision 810 determines whether the file access request is requesting a new folder. When the decision 810 determines that the file access request is seeking to request a new folder, then the requestor is prompted 812 for a new folder name. A decision 813 then determines whether a folder name has been received. When the decision 813 determines that a folder name has not yet been received, the file access request processing 800 waits for the folder name. Once the decision 813 determines that the folder name has been received, the new folder is created 814.

[0062] Alternatively, when the decision 810 determines that the file access request does not desire to create a new folder, then a decision 816 determines whether the file access request desires to download a file. When the decision 816 determines that the file access request desires to download a file, then the requested file is downloaded 818 to the requestor. On the other hand, when the decision 816 determines that the file access request does not desire to download a file, then a decision 820 determines whether the file access request desires to upload a file. When the decision 820 determines that the file access request does desire to upload a file, then the requested file is uploaded 822 to the requestor. Alternatively, when the decision 820 determines that the file access request does not desire to upload a file, then additional types of file access requests could be processed, although none are shown in FIG. 8A. Accordingly, following the decision 820 when the file access request does not desire to upload a file (and no additional types of file access requests are supported), then the file access request processing 800 is complete and ends with no file access operation having been performed. Following the blocks 808, 814, 818 and 822, the file access request processing 800 is also complete and ends but does so with the requested file access having been performed.

[0063] Furthermore, when the decision 802 determines that a server has not already been discovered, then the file access request processing 800 performs the processing shown in FIG. 8B. In this case, a list of available servers is initially discovered 824. Then, a decision 826 awaits the selection of one of the available servers by the requestor. Once the decision 826 determines that a server selection has been received, then share information for the selected server is retrieved 828. In one embodiment, the share information identifies those of the folders stored on the selected server that are able to be shared with third parties, such as remote requestors. A decision 830 then determines whether the information about the server should be made permanent. When the decision 830 determines that the information about the server should be made permanent, then the server information is saved 832. By saving the server information, the server is made an "available server" such that discovery of the availability of the server is not needed with subsequent logins to the system. On the other hand, when the decision 830 determines that the information about the server should not be made permanent, then the block 832 is bypassed. In any case, following the block 830 when the server information is not to be made permanent, as well as following the block 832 when the server information is to be made permanent, the processing to discover a server is complete and thus the file access request processing 800 returns to repeat the decision 802 and subsequent operations.

[0064] FIGs. 9A – 9C are flow diagrams of web resource request processing 900 according to one embodiment of the invention. The web resource request processing 900 is, for example, performed by an intermediary server, such as the intermediary server 108 illustrated in FIG. 1A or the intermediary server 158 illustrated in FIG. 1B. The web resource request processing 900 is performed to process a web resource request.

[0065] Initially, the host name for the appropriate remote server is obtained 902. In one embodiment, the host name can be obtained from storage. Here, the storage can, for example, be the data storage device 214 illustrated in FIG. 2A. In another embodiment, the host name can be obtained from the URL associated with the web resource request. After the host name for the appropriate remote server is obtained 902, a host name lookup is performed 904 to obtain an IP address of the appropriate remote server. A connection is then opened 906 (or maintained if

already opened) to the remote server. Next, a secure handshake is performed 908 between the intermediary server and the remote server as needed. Any "cookies" associated with the obtained host name are then obtained 910. Following the operation 910, the pre-processing of the web resource request at the intermediary server is complete and the request is now able to be forwarded to the remote server. At this point, the request for the web resource with associated "cookies" is sent 912 to the remote server.

[0066] A decision 914 then determines whether a response has been received. When the decision 914 determines that a response has not yet been received, the web resource request processing 900 awaits such a response. Once the decision 914 determines that a response has been received, then a decision 916 determines whether "cookies" are present in the response. When the decision 916 determines that "cookies" are present in the response, then the "cookies" are extracted 918 from the response. The extracted "cookies" are then saved 920. Typically, the "cookies" are stored in central storage provided within the intermediary server or other storage associated or coupled to the intermediary server. Following the operation 920, as well as following the decision 916 when it is determined that "cookies" are not present in the response, URLs within headers of the response are modified 922.

[0067] A decision 924 then determines whether the response is of a type that is to be modified. Here, in general, a response can be of a variety of forms such as HTML, graphics, .pdf, MPEG, or other formats. When the decision 924 determines that the response is of a type that cannot be modified (e.g., graphics), then the response can be immediately sent (or forwarded) 926 to the requestor. Then, a decision 928 determines whether the response is completed. When the decision 928 determines that the response is completed, then the web resource request processing 900 returns to repeat the decision 914 and subsequent operations so that additional web resource requests can be processed. Alternatively, when the decision 928 determines that so far only a portion of the response has been sent to the requestor, the web resource request processing 900 returns to repeat the decision 914 and subsequent operations or the like so that subsequent portions of the response can be similarly processed.

[0068] On the other hand, when the decision 924 determines that the response is of a type that can be modified (e.g., HTML), then the response is processed to



modify the response before returning it to the requestor. The processing illustrated in FIG. 9C represents one embodiment of processing that can be performed to modify the response. In particular, a decision 932 determines whether a toolbar is desired. The intermediary server can be configured to always, sometimes or never insert the toolbar. The toolbar can be standardized or customizable by the intermediary server. When the decision 932 determines that a toolbar is desired, the toolbar HTML is inserted into the response. The toolbar that is produced by the toolbar HTML can provide controls or content that are added to the resulting response so as to facilitate features or functionality provided by the intermediary server.

[0069] Next, certain URLs within an HTML portion of the response can be modified 936. In one embodiment, the modifications to the certain URLs can be achieved by modifying the host name portion of URLs within certain tags of the resulting HTML. In another embodiment, the modifications to the certain URLs can be achieved by adding suffixes to the certain URLs. The suffixes thus serve to allow the URLs to carry additional information. Further, certain URLs provided or produced by scripting language portions within the resulting HTML can be modified 938. Examples of scripting languages include JavaScript and VBScript. In one embodiment, a host name portion of the certain URLs provided or produced by scripting language portions within the resulting HTML are modified 938. In another embodiment, the certain URLs provided or produced by scripting language portions are modified 938 to include suffixes which carry supplemental information. Additional details on modifying scripting language portions is provided below with reference to FIGs. 13A and 13B. Thereafter, the modified response is sent 940 to the requestor.

[0070] A decision 942 then determines whether the request has been completed. When the decision 942 determines that the request has been completed, then the web resource request processing 900 is complete and ends. On the other hand, when the decision 942 determines that the request is not yet completed, then the web resource request processing 900 returns to repeat the decision 914 and subsequent operations so that remaining portions of the response can be similarly processed upon being received. The web resource request processing 900 can thus operate to process a single response to a resource request

in multiple pieces or blocks of data. In such a case, the web resource request processing 900 can process a response from a remote server as it arrives so that responsiveness to the requestor is not hindered. In this regard, the web resource request processing 900 causes the operations 914 – 942 to be repeated for each piece or block of data associated with a response.

[0071] FIG. 10 illustrates a diagram of an information retrieval system 1000 according to one embodiment of the invention. The information retrieval system 1000 is generally similar to the information retrieval system 100 of FIG. 1A or the information retrieval system 150 of FIG. 1B. The operation of the information retrieval system 1000 is discussed below with reference to a representative example which illustrates its operation according to one embodiment. The information retrieval system 1000 includes a client 1002, an intermediary server 1004 with a data store 1006, and a remote server 1008. It is assumed that the request processing 300 of FIG. 3 has already been performed and that the requestor is permitted to access the requested resource in the manner sought.

[0072] The representative example pertains to a secure request which can be initiated by the user selecting a hyperlink in a displayed webpage in the content of a web browsing request. The selected hyperlink is assumed to be

`https://secure.danastreet.com/quote/msft:danainfo:host=www.xyz.com`

where “https” is the protocol which uses Secure Socket Layer (SSL), “secure.danastreet.com” is the host name with “danastreet.com” being a domain and “secure” being a subdomain, “/quote/msft” being a path to the particular resource being requested by selection of the hyperlink, “danainfo” is a keyword, and “www.xyz.com” is the host where the requested resource resides. Hence, the domain name lookup of the host name “secure.danastreet.com” is resolved to the IP address of danastreet.com, which is the intermediary server 1004 for this example. The request is then sent from the client 1002 to the intermediary server 1004. The request is, for example, as follows:

GET: /quote/msft:danainfo:host=www.xyz.com HTTP/1.0

Host: secure.danastreet.com

Cookie: DSID = 123xyzzbc

Other information can also be included within the request such as additional cookies, encoding-accepted, etc. The cookie is, in this example, a session cookie (session identifier) and is used in determining whether the client 1002 is authorized for use with the intermediary server 1004.

[0073] In the case of a secure request, the host name within the request is not able to directly identify the remote server 1008 where the request is eventually to be delivered. However, the host name for the remote server 1008 is obtained from information provided with the request. More particularly, the information (i.e., host variable) is provided as a suffix with the request. In this example, the suffix includes the information that the host name of the remote server 1008 is "www.xyz.com". Once the appropriate host name has been obtained, a domain name lookup on the host name ("www.xyz.com") is performed. Next, a connection from the intermediary server 1004 and the remote server 1008 is opened (or maintained if already opened), and secure handshaking is optionally performed. Any cookies in the data store 1006 associated with the host name and the requestor can then be obtained. Next, a request by the intermediary server 1004 is sent to the remote server 1008. The request is, for example, as follows:

```
GET: /quote/msft HTTP/1.0
Host: www.xyz.com
Cookie: xyzUserID = sam
```

Other information can also be included within the request. Note that the cookie provided with the original request pertained to the intermediary server 1004 and thus is not forwarded with the request to the remote server 1008.

[0074] The remote server 1008 receives the request and returns a response header and some or all of the content of the requested resource. An exemplary response can have the following format:

```
HTTP/1.0 200 OK
Set-cookie: xyzuserID = Samual, expires = 01-Jul-2002
Content-type: text/html
Content-length: 2000
Location: https://www.xyz.com/quote/msft
<HTML>
```

\* \*

</HTML>

Since the response included a "cookie" to be set, the set-cookie command is removed from the response and then saved in the data store 1006. Next, to the extent they are present, the URLs within the headers are modified to point to the intermediary server 1004. In this example, the location header includes a full path (including host name), namely, https://www.xyz.com/quote/msft, which is thus modified to

https://secure.danastreet.com/quote/msft:danainfo:host=www.xyz.com,SSL. In this example, not only are the host names modified but variables are also added to the end (i.e., suffix) of the URL. The variable information added is an indication of the host server having the requested resource and an SSL indicator. With this example, the relative URLs need to be modified to include the variable information ("danainfo:host=www.xyz.com") at the end of the relative URLs. The host names for the relative URLs are properly provided by the browser application operating on the client 1002 which causes the current host name ("secure.danastreet.com") to be used for such paths. If desired, a toolbar can be inserted into the HTML data to facilitate operations or functions supported by the intermediary server 1004. Still further, the URLs within certain tags within the resulting HTML or those produced by scripting languages are modified to point to the intermediary server 1004.

[0075] For example, if the HTML data included the following hyperlink:

<a ref = https://www.xyz.com/quote/msft>

then the hyperlink would be modified to the following:

<a ref=https://secure.danastreet.com  
/quote/msft:danainfo:host=www.xyz.com,SSL>.

Also, if the HTML data included the following relative hyperlink:

<a ref = a.html>

then the hyperlink would be modified to the following:

<a ref = a.html:danainfo:host=www.xyz.com,SSL>.

It should be noted that the variable information provided at the end (i.e., suffix) of the URLs need not be at the actual end. Here, suffix is used to generally indicate to

the right of the domain name. Indeed, the variable information can be placed in a variety of different locations in a URL (even to the left of the domain name). For example, if the original hyperlink itself has variables such as following the characters "?" or "#", then the variable information ("danainfo:host=www.xyz.com") can, in one example, be placed before the character "?" or "#" indicating the original variables. For example, if the HTML data included the following hyperlink:

```
<a ref = https://www.xyz.com/quote/msft?color=red>
```

then the hyperlink would be modified to the following:

```
<a ref=https://secure.danastreet.com  
/quote/msft:danainfo:host=www.xyz.com?color=red>.
```

Also, if the HTML data included the following relative hyperlink:

```
<a ref = a.html?x=1234>
```

then the hyperlink would be modified to the following:

```
<a ref = a.html:danainfo:host=www.xyz.com?x=1234>.
```

As still another example, if the HTML data included the following relative hyperlink:

```
<a ref = a.html, port=1234>
```

then the hyperlink would be modified to the following:

```
<a ref = a.html:danainfo:host=www.xyz.com, port=1234>.
```

[0076] FIG. 11 is a flow diagram of URL modification processing 1100 according to one embodiment of the invention. The URL modification processing 1100 is, for example, processing performed by operation 936 of FIG. 9C. The URL modification processing 1100 can, for example, be performed by the content transformer 216 illustrated in FIG. 2A or the content transformer 278 illustrated in FIG. 2B.

[0077] The URL modification processing 1100 begins by selecting 1102 a target URL within an HTML portion of the response (e.g., webpage). Typically, one or more target URLs are previously identified by scanning the HTML data. Then, a decision 1104 determines whether the target URL is a relative URL. A relative URL inherits the characteristics of its source URL. The source URL is the URL associated with the webpage (including the resulting HTML) that includes the target URL. When the decision 1104 determines that the target URL is a relative URL, then the hostname and/or port suffix from the source URL are appended 1106 to the target URL.

[0078] Alternatively, when the decision 1104 determines that the target URL is not a relative URL, then a decision 1108 determines whether the target URL is associated with a secure request (e.g., HTTPS). When the decision 1108 determines that the request for the target URL is a secure request, then a secure indicator (e.g., HTTPS) is added 1110 to the target URL. On the other hand, if the decision 1108 determines that the target URL is not associated with a secure request, the operation 1110 is bypassed.

[0079] Following the operation 1110 as well as directly following the decision 1108 when the target URL is not associated with a secure request, then the host name provided with the target URL is added 1112 elsewhere to the target URL. For example, the host name provided with the target URL can be appended to the target URL. Then, the original host name provided with the target URL is replaced 1114 with a predetermined host name. In other words, the host name originally provided for the target URL is effectively rewritten such that the original host name is replaced with the predetermined host name, but the original host name remains part of the target URL. For example, the predetermined host name is the host name for the appropriate intermediary server.

[0080] Next, a decision 1116 determines whether a port number is specified in the target URL. When the decision 1116 determines that a port number is specified in the target URL, then a port number suffix is added 1118 to the target URL. The port number originally specified in the target URL following the host name is removed 1120.

[0081] Following the operation 1120, the URL modification processing 1100 performs a decision 1122. Additionally, when the decision 1116 determines that a port number is not specified in the target URL, no port number processing is needed so the decision 1122 is then performed. The decision 1122 determines whether more target URLs are to be processed. As previously noted, these target URLs have been previously identified by scanning the resulting HTML data. When the decision 1122 determines that there are more target URLs, then the URL modification processing 1100 returns to repeat the operation 1102 and subsequent operations so that additional target URLs can be processed. Alternatively, when the decision 1122 determines that there are no more target URLs, then the URL modification processing 1100 is complete and ends.

[0082] FIG. 12 is a flow diagram of a script modification processing 1200 according to one embodiment of the invention. The script modification processing 1200 is, for example, performed by operation 938 illustrated in FIG. 9C. In general, the script modification processing 1200 operates to modify script portions within the resulting HTML.

[0083] The script modification processing 1200 initially scans 1202 the HTML data (e.g., of the resulting HTML) for a <script> tag. A decision 1204 then determines whether a script has been found. When the decision 1204 determines that a script has not been found, then a decision 1206 determines whether there is more HTML data to be scanned. When the decision 1206 determines that there is more HTML data to be scanned, then the script modification processing 1200 returns to repeat the operation 1202 and subsequent operations. Alternatively, when the decision 1206 determines that there is no more HTML data to be scanned, the script modification processing 1200 is complete and ends.

[0084] On the other hand, when the decision 1204 determines that a script has been found, then the script is searched 1208 to locate text strings "http://" or "https://" followed by a host name. A decision 1210 then determines whether a URL host name has been found by the searching 1208 of the script. When the decision 1210 determines that a URL host name has not been found, then a decision 1212 determines whether the end of the script has been reached. When the decision 1212 determines that the end of the script has not yet been reached, then the script modification processing 1200 returns to repeat the operation 1208 and subsequent operations. Alternatively, when the decision 1212 determines that the end of the script has been reached, then the script modification processing 1200 returns to repeat the operation 1202 and subsequent operations so that additional scripts can be found and processed.

[0085] On the other hand, when the decision 1210 determines that a URL host name has been found, then a rewritten host name is produced 1214. The host name provided within the script is then replaced 1216 with the rewritten host name. Following the operation 1216, the script modification processing 1200 returns to repeat the operation 1208 and subsequent operations so that additional host names within the script can be similarly processed.

[0086] FIGs. 13A and 13B are flow diagrams of a script modification processing 1300 according to another embodiment of the invention. The script modification processing 1300 is, for example, performed by operation 938 illustrated in FIG. 9C. In general, the script modification processing 1300 operates to modify script portions within the resulting HTML.

[0087] The script modification processing 1300 initially scans 1301 the HTML data (e.g., of the resulting HTML) for a <script> tag. A decision 1302 then determines whether a script has been found. When the decision 1302 determines that a script has been found, then the script is parsed 1304 to determine or locate predetermined properties and functions associated with the script. A decision 1306 then determines whether at least one property or function has been found in the script. When the decision 1306 determines that at least one property or function has been found, then the script modification processing 1300 continues such that the script is modified with respect to the properties or functions found within the script so that the script operates as intended even though the intermediary server is interposed between client devices and remote servers.

[0088] In particular, for each property or function found within the script, the processing is as follows. A decision 1308 determines whether a selected property or function found within the script pertains to a read of a cookie property. When the decision 1308 determines that the identified property or function does pertain to a read of a cookie property, then the read of the cookie property is replaced 1310 with a get\_cookies function call. Alternatively, when the decision 1308 determines that the identified property or function is not a read of a cookie property, as well as following the operation 1310, a decision 1312 determines whether the identified property or function pertains to a write to a cookie property. When the decision 1312 determines that the identified property or function does pertain to a write to a cookie property, the write to the cookie property is replaced 1314 with a set\_cookies functions call.

[0089] On the other hand, when the decision 1312 determines that the identified property or function is not associated with a write to a cookie property, as well as following the operation 1314, a decision 1316 determines whether the identified property or function pertains to a write to a property that initiates a request. When the decision 1316 determines that the identified property or function



does pertain to a write to a property that initiates a request, then the write to the property that initiates (causes) a request is replaced 1318 with a set\_URL function call. Alternatively, when the decision 1316 determines that the identified property or function does not pertain to a write to a property that initiates a request, as well as following the operation 1318, a decision 1320 determines whether the identified property or function pertains to a read from a property that returns a URL. When the decision 1320 determines that the identified property or function does pertain to a read from a property that returns a URL, then the read from a property that returns a URL is replaced 1322 with an appropriate string.

[0090] Furthermore, following the decision 1320 when the identified property or function does not pertain to a read from a property that returns a URL, as well as following the operation 1322, a decision 1324 determines whether more properties or functions were found in the script that still need to be processed. When additional properties or functions have been found and need processing, the script modification processing 1300 returns to repeat the decision 1308 and subsequent operations so that the additional properties or functions can be similarly processed. On the other hand, when the decision 1324 determines that all the properties or functions that have been found within the script have been processed, then the script modification processing 1300 performs a decision 1326. The decision 1326 is also performed when the decision 1302 determines that a script has not been found. The decision 1326 determines whether there is more HTML data to be scanned. When the decision 1326 determines that there is more HTML data to be scanned, then the script modification processing 1300 returns to repeat the operation 1301 and subsequent operations. Alternatively, when the decision 1326 determines that there is no more HTML data to be scanned, the script modification processing 1300 is complete and ends.

[0091] Representative examples of a get\_cookies function, a set\_cookies function, a set\_URL function, and string substitution are provided below. These examples are provided to assist understanding and thus should not be deemed restrictions on any aspect of the invention. The following examples use JavaScript as the scripting language.

[0092] A first example with respect to the get\_cookies function and operation 1310 is as follows. In this example, the script includes a script instruction

`var c = document.cookie;`

which assigns the cookies associated with the document (page) to the variable c.

This script instruction would be replaced with

`var c = get_cookies ("othercookie=abc");`

which assigns the cookies present on the intermediary server for the particular domain of the document (page) and the particular user (e.g., "othercookie=abc"). In addition, the `get_cookies` function takes the cookies from the intermediary server as its argument and adds to it other cookies that are set by the script.

[0093] A second example with respect to the `set_cookies` function and operation 1314 is as follows. In this example, the script includes a script instruction

`document.cookie = "selection=ijk; expires= ...";`

which stores the cookies associated with the document (page) in the browser. This script instruction (statement) is replaced with

`document.cookie = set_cookie ("<domain>", "selection=ijk; expires= ...");`

which stores the cookies associated with the document (page) in the browser and also to the intermediary server. The `set_cookie` function includes two arguments. The first argument identifies the domain of the page having the script. The second argument is the value that was originally being assigned to the `document.cookie` property. The `set_cookie` function combines these two arguments and sets a cookie called `servercookieX` with no expiration, where X represents a distinguishing numeric value. The browser will cause this cookie to be sent to the intermediary server. The intermediary server can then incorporate the cookie into the cookie storage for the user. The cookie can also be used to expire an existing cookie in storage for the user. Once the cookie is stored at the intermediary server, the next page that the intermediary server returns will cause the `servercookieX` to expire because it is no longer needed. Any calls to the `set_cookie` function will also append any cookie values provided within the `servercookieX`.

[0094] To further illustrate, consider the following example where a page from `www.xyz.com` has the following script:

`document.cookie = "a=b";`

`var x = document.cookie;`

Assume also the www.xyz.com server has previously returned a cookie to the intermediary server that has a name "id1" with a value "sam". The code above will be transformed into:

```
document.cookie = set_cookie ("www.xyz.com", "a=b");  
var x = get_cookie ("id1=sam");
```

The first line will cause a cookie "servercookie0" to be set that has the value "a=b ~ domain=www.xyz.com", hence the whole cookie will be:

```
servercookie0 = a=b ~ domain=www.xyz.com.
```

Note that the domain part of the servercookie0 is used purely by the intermediary server so that it knows which domain is setting the cookie. The second line calls the get\_cookies function which takes its first argument (filled in by the intermediary server while the script was rewritten) and examines all servercookie0's cookies at the browser. It concatenates the first argument together with any servercookieX cookies, and returns the result:

```
id1=sam; a=b.
```

Note, this is the same result that would have been returned from the original page had it not been rewritten.

[0095] A third example with respect to the set\_URL function and operation 1318 is as follows. The set\_URL function operates to modify properties that cause a request. In this example, the script includes a script instruction

```
document.location = "http://www.xyz.com/foo.html";
```

which directs the browser to a new page. Such a script instruction can be replaced with

```
document.location = set_URL( "", "http://www.xyz.com/foo.html");
```

The set\_URL function call takes two arguments. The first argument is filled in by the intermediary server while the script is being rewritten and contains any parameters that would normally be provided in a suffix (e.g., "danainfo:") to follow a URL. It is not always needed, as will be explained below. The second argument is the URL, though it could actually be a script expression (e.g., function call) that assembles or returns a URL.

[0096] The set\_URL function examines the URL being set and rewrites it to be of a form that will direct the browser to the intermediary server. As noted above, the modification to URLs can be achieved with various techniques.

[0097] If the page is using the host name modification technique, then relative URLs do not need to be modified since the host name encodes the necessary information. If the URL is a full URL, then the set\_URL function has all of the information it needs to convert the URL. For example, a suffix (e.g., ":danalInfo:host=xxx") can be appended to the URL. Thus, if the page that the script appears on is using the host name modification technique, the first argument is not needed by the set\_URL function.

[0098] Alternatively, if the page upon which the script is present is using the URL suffix technique, then a relative URL that is passed to the set\_URL function needs to have the same suffix applied to it. In this case, the intermediary server will insert, as the first argument to the set\_URL function, any arguments that need to be passed in the suffix. For example, if the URL of the page is:

`https://secure.danastreet.com/quote/msft:danalInfo:host=www.xyz.com`

and a script instruction on the page includes:

`document.location = "/quote/ibm";`

then the rewritten script instruction would look like:

`document.location = set_URL("Host=www.xyz.com", "/quote/ibm");`

and the returned result from the set\_URL function would be:

`/quote/ibm:danalInfo:host=www.xyz.com`

which would result in a request from the browser for:

`https://secure.danastreet.com/quote/ibm:danalInfo:host=www.xyz.com.`

Alternatively, if the script instruction were instead:

`document.location = "https://www.abc.com/info/msft";`

then the rewritten script instruction would look like:

`document.location = set_URL("Host=www.xyz.com",`

`"https://www.abc.com/info/msft");`

and the returned result from the set\_URL function would be:

`https://secure.danastreet.com/info/msft:danalInfo:host=www.abc.com.`

Note that, in this case, the first argument to the set\_URL function is not needed because the second argument is a full URL and contains all of the information necessary to construct the final URL.

[0099] It should be noted that there are many functions or properties that, when written to, can cause the browser to fetch a URL. Some examples include:

```
window.open('url', ...)
form.action = 'url';
document.location = 'url';
document.location.replace('url');
image.src = 'url';
```

[00100] A fourth example with respect to the string substitution and operation 1322 is as follows. The string substitution operates to modify properties that return a URL. Here, script instructions that read from a property that return a URL are replaced with a constant string. In this example, if the script includes

```
var url = document.location;
```

such would be replaced by:

```
var url = "http://www.yahoo.com/foo.html";
```

This operation serves to ensure that any script examining its environment would not be confused by the fact that the actual URL of the page is different from what it expects. Note that there is more than one property that may need to be modified.

Some examples of properties that can be so modified include:

```
document.location (returns full URL)
```

```
document.domain (returns just the hostname part of URL).
```

[00101] FIG. 14 is a flow diagram of email request processing 1400 according to one embodiment of the invention. The email request processing 1400 is, for example, suitable for use as the email request processing performed at block 628 of FIG. 6.

[00102] The email request processing 1400 initially accepts 1402 a secure connection with a mail client. Here, the secure connection between the mail client and the intermediary server that is being accepted 1402 can, for example, be made secure through use of a Secure Socket Layer (SSL). Next, the requestor is prompted 1404 for authentication. Typically, the requestor is prompted 1404 to enter at least a password that can be used to authenticate the requestor. A decision 1406 then determines whether a password has been received. Typically, but not necessarily, the password being received is encoded in some manner. For example, base-64 encoding is often utilized. When the decision 1406 determines

that a password has been received, then the password can be separated 1408 into a mail server password and an authentication server password. As an example, the received password can include both the mail server password and the authentication server password separated by a password separator.

[00103] Next, the email server attempts to verify 1410 the mail server password. At about the same time, the authentication server password can attempt to be verified 1412 with the authentication server. Next, a decision 1414 determines whether both of the verifications of blocks 1410 and 1412 have been successful. When the decision 1414 determines that both of the verifications have been successful, then a hashed version of the password is stored 1416. Then, the mail operation processing 1418 associated with the email request is performed. On the other hand, when the decision 1414 determines that both of the verifications of blocks 1410 and 1412 are not successful, then the email request is denied 1420. Following the operations 1418 and 1420, the email request processing 1400 is complete and ends.

[00104] FIG. 15 is a flow diagram of mail operation processing 1500 according to one embodiment of the invention. The mail operation processing 1500 is, for example, one example of processing that can be performed by the block 1418 illustrated in FIG. 14.

[00105] The mail operation processing 1500 begins with a decision 1502 that determines whether the connection has timed-out or closed. Here, the connection refers to the secure connection between the mail client and the intermediary server. When the decision 1502 determines that a secure connection has timed-out or closed, then email access to the mail server is denied 1504. Hence, following the block 1504, the mail operation processing is complete and ends when the secure connection has timed-out or closed. However, the processing could continue to return a login page to the requestor to force the requestor to login and be authenticated in order to gain access to the mail server.

[00106] On the other hand, when the decision 1502 determines that an existing connection has not timed-out or closed, then a decision 1506 determines whether a command from a mail client has been received. When the decision 1506 determines that a command from a mail client has not been received, then the mail operation processing 1500 returns to repeat the decision 1502 and subsequent

operations until a command from the mail client has been received or until the connection has timed-out or otherwise closed.

[00107] Once the decision 1506 determines that a command from a mail client has been received, then the command is forwarded 1508 to the mail server. Next, a decision 1510 determines whether a response has been received from the mail server. When the decision 1510 determines that a response has not yet been received from the mail server, then the mail operation processing 1500 awaits such a response. Once the decision 1510 determines that a response has been received, then certain Universal Resource Locators (URLs) within the response are modified 1512. For example, as part of the content transformation, links or URLs are able to be modified to redirect the links through the intermediary server. Next, the response is sent 1514 to the mail client. Here, the response is sent to the mail client using the connection that exists between the mail client and the intermediary server. Following the block 1514, the mail operation processing 1500 returns to repeat the decision 1502 and subsequent operations so that additional commands can be processed with respect to the mail server.

[00108] FIG. 16 is a flow diagram of authentication processing 1600 according to one embodiment of the invention. The authentication processing 1600 represents one embodiment of the block 1412 illustrated in FIG. 14. In this embodiment, the intermediary server is able to bypass or avoid actual verification of a password with the authentication server when certain conditions are met. By doing so, the authentication can, in many cases, be performed very quickly and without the need to burden or annoy requestors.

[00109] The authentication processing 1600 begins with a decision 1602 that determines whether a stored hashed password is available. When a hashed password is previously stored (operation 1416 of FIG. 14), the hashed password can later be retrieved and used in this regard. Hence, when the decision 1602 determines that the stored hashed password is available, then the stored hashed password, a time last authorized and a time last used password are retrieved 1604. Typically, these values are stored in the data store associated with the intermediary server and are stored values that are particular to the requestor.

[00110] Next, a decision 1606 determines whether a hash of the received password equals the stored hashed password. When the decision 1606 determines

that the hash of the received password is equal to the stored hashed password, then the requestor is, in effect, authenticated, because earlier in the session they entered the proper password that was then authenticated. Further, a decision 1610 determines whether the time since the time last authorized is greater than a maximum session duration. Here, the variable indicating the duration of time that has expired since the time last authorized is compared to the maximum session duration. Typically, the maximum session duration is set by the requestor or by the system administrator of the intermediary server.

[00111] In any case, when the decision 1610 determines that the time since the time last authorized does not exceed the maximum session duration, then a decision 1612 determines whether the time since the time last used password exceeds a maximum idle time. Here, the variable indicating the duration of time that has expired since the time last used password is compared to the maximum idle time. When the decision 1612 determines that the time since last used the password does not exceed the maximum idle time, then authentication 1614 by the authentication server is deemed successful without having to interact with the authentication server. Hence the authentication with respect to the authorization server is able to be bypassed when the hash of the received password equals the stored hash password, provided the time since last authorized does not exceed the maximum session duration, and further provided the time since last used the password does not exceed the maximum idle time.

[00112] On the other hand, the password is verified 1608 with the authentication server when the special conditions do not exist. For example, when the decision 1602 determines that the stored hash password is not available, then the verification 1608 with the authentication server is performed. Likewise, when the decision 1606 determines that the hash of the received password is not equal to the stored hash password, then the verification 1608 of the password with the authentication server also needs to be performed. Still further, when the decision 1610 determines that the time since last authorized exceeds the maximum session duration or when the decision 1612 determines that the time since last used the password exceeds the maximum idle time, then the password needs to be verified 1608 with the authentication server.



[00113] Following the operations 1608 and 1614, the authentication processing 1600 returns to perform other processing, namely, returns to the operation 1414 illustrated in FIG. 14. Hence, when the verification 1608 is able to be bypassed because the above-mentioned special conditions exist, the authorization processing is greatly simplified and often avoids the need to perform complicated authentication processing with respect to an authentication server or to prompt a requestor for authentication information.

[00114] FIGs. 17A and 17B illustrate an example of a computer system that may be used in accordance with the invention. The computer system can, for example, correspond to any of the client machines, the intermediary server, or the remote or private servers. FIG. 17A shows a computer system 1721 that includes a display 1723, screen 1725, cabinet 1727, keyboard 1729, and mouse 1731. Mouse 1731 may have one or more buttons for interacting with a graphical user interface. The cabinet 1727 houses a removable medium (e.g., CD-ROM) drive 1733, system memory and a hard drive (see FIG. 17B) which may be utilized to store and retrieve software programs incorporating computer code that implements the invention, data for use with the invention, and the like. Although CD-ROM 1735 is shown as an exemplary computer readable storage medium, other computer readable storage media including floppy disk, tape, DVD, flash memory, system memory, and hard drive may be utilized. Additionally, a data signal embodied in a carrier wave (e.g., in a network including the Internet) may be the computer readable storage medium. In one implementation, an operating system for the computer system 1721 is provided in the system memory, the hard drive, the CD-ROM 1735 or other computer readable storage medium and serves to incorporate the computer code that implements the invention.

[00115] FIG. 17B shows a system block diagram of the computer system 1721 used to perform the processing of an embodiment of the invention. As in FIG. 17A, the computer system 1721 includes monitor 1723, keyboard 1729, and mouse 1731. The computer system 1721 further includes subsystems such as a central processor 1751, system memory 1753, fixed storage 1755 (e.g., hard drive), removable storage 1757 (e.g., compact disk), display adapter 1759, sound card 1761, speakers 1763, and network interface 1765. The central processor 1751 can, for example, execute computer program code (e.g., an operating system) to implement the

invention. An operating system is normally, but necessarily, resident in the system memory 1753 during its execution. Other computer systems suitable for use with the invention may include additional or fewer subsystems. For example, another computer system could include more than one processor 1751 (*i.e.*, a multi-processor system) or a cache memory.

[00116] The system bus architecture of computer system 1721 is represented by arrows 1767. However, these arrows are illustrative of any interconnection scheme serving to link the subsystems. For example, a local bus could be utilized to connect the central processor to the system memory and display adapter. The computer system 1721 shown in FIG. 17A is but an example of a computer system suitable for use with the invention. Other computer architectures having different configurations of subsystems may also be utilized.

[00117] Although the above-described embodiments refer to the use of a single intermediary server within an information retrieval system, it should be recognized that the information retrieval system can also include a plurality of intermediary servers. The various intermediary servers can individually receive requests from client machines and forward them to the appropriate servers and return responses back through the intermediary server to the client machine. By having multiple servers, not only can additional processing power be obtained, but load balancing, fault tolerance and localization issues can also be addressed.

[00118] The various aspects, features, embodiments or implementations of the invention described above can be used alone or in various combinations.

[00119] The invention is preferably implemented in software, but can be implemented in hardware or a combination of hardware and software. The invention can also be embodied as computer readable code on a computer readable medium. The computer readable medium is any data storage device that can store data which can thereafter be read by a computer system. Examples of the computer readable medium include read-only memory, random-access memory, CD-ROMs, DVDs, magnetic tape, optical data storage devices, and carrier waves. The computer readable medium can also be distributed over network-coupled computer systems so that the computer readable code is stored and executed in a distributed fashion.

[00120] The advantages of the invention are numerous. Different embodiments or implementations may yield one or more of the following advantages. One advantage of the invention is that an intermediary server can be interposed between remote servers and clients to facilitate secure access to email resources. Another advantage of the invention is that requestors seeking access to resources on private networks can be authenticated through use of native authentication provided by the private network. Still another advantage of the invention is that secure remote access to private networks can be easily provided for authorized persons at reasonable costs.

[00121] The many features and advantages of the present invention are apparent from the written description and, thus, it is intended by the appended claims to cover all such features and advantages of the invention. Further, since numerous modifications and changes will readily occur to those skilled in the art, it is not desired to limit the invention to the exact construction and operation as illustrated and described. Hence, all suitable modifications and equivalents may be resorted to as falling within the scope of the invention.

*What is claimed is:*